Tampereen yliopisto

Jukka Yrjänäinen

NEUROVERKKOKIIHDYTYSTÄ KÄYTTÄVÄN KONENÄKÖSOVELLUKSEN TOTEUTUS

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Helmikuu *2019*

TIIVISTELMÄ

JUKKA YRJÄNÄINEN: Neuroverkkokiihdytystä käyttävän konenäkösovelluksen toteutus
Tampereen teknillinen yliopisto
Kandidaatintyö, 28 sivua
Helmikuu 2019
Sähkötekniikan kandidaatin tutkinto-ohjelma
Pääaine: Signaalinkäsittely
Tarkastaja: Apulaisprofessori Heikki Huttunen

Avainsanat: konenäkö, koneoppiminen, neuroverkot

Tässä työssä tarkastellaan syviä neuroverkkoja käyttävän konenäkösovelluksen toteuttamista. Tutkimuksen käytettiin *Raspberry Pi* pienoistietokone ympäristöä, johon on liitetty neuroverkkolaskentaa nopeuttava erillinen laite: *Movidius Neural Computing Stick*. Työn tavoitteena oli selvittää, miten kyseinen järjestelmä soveltuu konenäkötehtäviin erityisesti tuotekehitys- ja tutkimustyön näkökulmasta.

Tutkimuksen alussa esitellään yleisesti neuroverkkolaskennan periaatteita ja käytettyjä ohjelmistoympäristöjä. Seuraavaksi tutustutaan laskennan nopeuttamiseen suunniteltuun neuroverkkokiihdyttimeen. Tämän jälkeen tarkastellaan lähemmin konenäkösovelluksen käytännön toteuttamiseen käytetyn laitteiston ja ohjelmiston keskeisiä ominaisuuksia ja esitellään työssä kehitettyjen ohjelmistojen keskeiset suunnitteluratkaisut. Lopuksi raportoidaan tehtyjen suorituskykymittausten tulokset ja niistä vedetyt johtopäätökset.

Mittauksista nähdään, että tutkittu järjestelmä on riittävän suorituskykyinen reaaliaikaisiin konenäkösovelluksiin. Käytetyt ohjelmointiympäristöt ovat joustavia ja sopivat erilaisten ratkaisujen nopeaan kehitykseen ja testaamiseen. Yhteenvetona tutkimuksen tuloksista voidaan sanoa, että työssä käytetty laite- ja ohjelmistoympäristö soveltuu varsin hyvin konenäköön ja neuroverkkoihin liittyvään tuotekehitys- ja tutkimustyöhön.

ABSTRACT

JUKKA YRJÄNÄINEN: Implementation of Machine Vision Application Utilizing Neural Computation Acceleration
Tampere University of Technology
Bachelor of Science Thesis, 28 pages
February 2019
Bachelor of Science Program in Electrical Engineering
Major: Signal Processing
Examiner: Associate Professor Heikki Huttunen

Keywords: machine vision, machine learning, neural networks

This thesis work studies the implementation of the machine vision application with deep neural networks. *Raspberry Pi* single board computer with *Movidius Neural Computing Stick* accelerator was used as the platform for the study. The target of the work was to analyse how suitable this platform and related software environment are for machine vision research and development.

In the beginning of the work general principles of neural computation and related software frameworks are introduced. After that the acceleration of the neural networks is covered. Key features of the hardware and software used in the work are presented, followed by discussion of main architectural design choices used in applications developed during the work. Finally results of the performance measurements and conclusion are presented.

Measurements indicate that performance of the platform under study is at sufficient level for real time computer vision applications. The programming environment is flexible and well suited for rapid prototyping. As a conclusion it can be stated that the system analysed in this work is well suited for machine vision and neural network related research and development work.

ALKUSANAT

Tämä tutkielma on tehty Tampereen Teknillisen Yliopiston Signaalinkäsittelyn laitokselle osana tekniikan kandidaatin tutkintoa.

Haluan kiittää apulaisprofessori Heikki Huttusta hyvästä työn ohjauksesta ja apulaisprofessori Joni Kämäräistä hyvin toteutusta kandidaattiseminaarista.

Ylöjärvellä, 13.2.2019

Jukka Yrjänäinen

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

CNN	engl. Convolution Neural Network, konvoluutioneuroverkko.
CPU	engl. Central Processing Unit, tietokoneen keskusyksikkö.
DNN	engl. Deep Neural Network, syvä neuroverkko.
et al.	lat. et alii tai et alieae, ja muut.
GLI	engl. Global Interpreter Lock, CPython tulkin toiminnan rajoitus yhteen prosessoriytimeen.
GPU	engl. Graphics Processing Unit, grafiikan prosessointiyksikkö.
HDMI	engl. High Definition Media Interface, suuren tarkkuuden media liityntä.
ISP	engl. Image Signal Processor. Kuvasignaaliprosessori.
ILSVRC	engl. ImageNet Large Scale Visual Recognition Challenge. Kuvanluokittelukilpailu.
MIPI	Järjestö, joka määrittelee liityntä standardeja mobiililaitteisiin, esimerkiksi MIPI kamera liitäntä.
MNCS	engl. Movidius™ Neural Computing Stick, Intelin valmistama neuroverkkokiihdytin.
NCAPI	engl. Neural Computing Application Programing Interface, neuroverkkokiihdyttimen käyttämä ohjelmointirajapinta.
PC	engl. Personal Computer, henkilökohtainen tietokone.
RGB	engl. Red Green Blue, sininen vihreä punainen, värikuvan esitysmuoto.
RLU	engl. Rectified Linear Unit, eräs neuroverkon aktivaatiofunktio, rajoitettu lineaarinen yksikkö.
USB	engl. Universal Serial Bus, yleiskäyttöinen sarjamuotoinen tiedonsiirtoväylä.
VLIW	engl. Very-Long Instruction Word. Pitkän sanapituuden tietokonearkkitehtuuri.
VPU	engl. Visual Prosessing Unit. Visuaalinen prosessointiyksikkö.
SIMD	engl. Single Instruction Multiple Data. Rinnakkaisprosessointi tietokonearkkitehtuuri.
SDK	engl. Software Development Kit. Ohjelmistonkehitysympäristö.
SSD	engl. Single Shot Multibox Detector. Kohteiden paikantamiseen suunniteltu neuroverkkoarkkitehtuuri.

1. JOHDANTO

Tekoälyn ja koneoppimisen sovellukset ovat levinneet laajasti erilaisille sovellusalueille. Tekniikan kehittymisen myötä ratkaisut, jotka aiemmin ovat vaatineet runsaasti laskentaa ja tehokkaita tietokoneita ovat tänään mahdollisia pienissä kannettavissa laitteissa kuten esimerkiksi matkapuhelimissa, osana sulautettuja järjestelmiä tai puettavaa elektroniikkaa. Tämä teknologian kehityssuunta on avannut mahdollisuuksia myös suuremmille hajautetuille järjestelmille, jossa informaatiota prosessoidaan pienemmissä, vähän tehoa kuluttavissa laitteissa ennen kuin prosessoitu tieto lähetetään keskusjärjestelmään.

Alueen teknologian kehitys on nopeaa ja monet yritykset, yliopistot ja myös yksityiset hankkeet ovat toteuttaneet ratkaisuja, joilla tekoälysovelluksia voidaan kehittää ja tuotteistaa nopeasti. Saatavilla on useita erilaisia kehitysympäristöjä ja ohjelmakirjastoja, joilla sovelluskehittäjät pystyvät toteuttamaan suhteellisen pitkälle optimoituja ratkaisuja ilman, että tarvitaan tietoa käytetyn järjestelmän yksityiskohdista. Yleisesti käytetyillä ohjelmakirjastoilla tehtyjä toteutuksia on myös suhteellisen helppo siirtää järjestelmästä ja sovelluksesta toiseen. Tämä mahdollistaa sen, että sovelluskehittäjien ei välttämättä tarvitse olla tekoälyteknologian asiantuntijoita vaan he pystyvät käyttämään jo tehtyjä ratkaisuja tunnettuihin ongelmiin. Myös alueen asiantuntijat ja tutkijat hyötyvät näistä ympäristöistä, uusien tutkimustulosten testaaminen ja jakaminen tiedeyhteisössä on tehokasta ja tämä on osaltaan mahdollistanut alan viime vuosien nopean kehityksen. Nopeasti kehittyvällä teknologia-alueella on kuitenkin toisinaan haasteena järjestelmien keskeneräisyys, kattavan dokumentaation ja ohjeistuksen puute.

Tekoäly ja koneoppiminen on laaja alue, joka sisältää useita erilaisia menetelmiä ja lähestymistapoja. Syvät neuroverkot ovat yksi tämän hetken suosituimmista koneoppimisen menetelmistä ja niillä on saavutettu hyviä tuloksia ratkaistaessa ongelmia, jotka ovat perinteisesti olleet haastavia tekoälyjärjestelmille. Neuroverkkoja on käytetty esimerkiksi konenäkösovelluksissa, isojen tietomäärinen analyysissä, luonnollisen kielen käsittelyssä ja tietokoneohjelmissa, jotka pystyvät voittamaan parhaat ihmispelaajat monimutkaisissa strategiapeleissä kuten shakissa ja go:ssa. Syvien neuroverkkojen rakenne mahdollistaa myös niiden vaatiman laskennan tehokkaan toteuttamisen erilaisissa rinnakkaisprosessointiarkkitehtuureissa. Neuroverkkoja varten onkin kehitetty omia prosessoreita, mutta myös yleiskäyttöisempiä rinnakkaiseen laskentaan optimoituja ratkaisuja, kuten grafiikkakiihdyttimiä on käytetty laajasti. Luvussa 2 perehdytään neuroverkkoihin tarkemmin.

Konenäöllä tarkoitetaan yleisesti prosessointijärjestelmiä, jotka analysoivat kuva- tai videosignaaleja ja tuottavat tämän pohjalta informaatiota. Tässä työssä on tutkittu syviä

neuroverkkoja käyttävän konenäkösovelluksen toteutusta ulkoisella neuroverkkokiihdyttimellä varustetulla pienoistietokoneella. Tutkimuksen tavoitteena on ollut testata, kuinka kyseinen laitteisto ja sen ohjelmointiympäristö soveltuvat tämäntyyppisten sovellusten tutkimiseen ja kehittämiseen.

Lähtökohtaisesti pienoistietokone ja siihen liittyvä kiihdytin on kiinnostava yhdistelmä konenäkötutkimuksen ja -tuotekehityksen tarpeisiin. Järjestelmä on suhteellisen edullinen, joka mahdollistaa useiden laitteiden hankinnan ja esimerkiksi hajautettuun kameravalvontajärjestelmän kustannustehokkaan testaamisen. Laitteiston tehonkulutus on myös riittävän pieni, jotta sitä voidaan käyttää akkujen avulla. Lisäksi järjestelmä on pienikokoinen ja siitä voidaan tarvittaessa tehdä helposti siirrettävä. Nämä ominaisuudet mahdollistavat erilaisten sovelluksien ja algoritmien testaamisen realistisessa käyttöympäristössä. Soveltuvuutta rajoittavia tekijöitä ovat järjestelmän suorituskyky sekä kehitystyön tuottavuus, lähinnä ohjelmoinnin helppous ja yhteensopivuus yleisesti käytettyjen neuroverkkojen ohjelmointiympäristöjen kanssa.

Alustana tässä tutkimuksessa on käytetty Raspberry Pi 3 tietokonetta, johon on saatavilla erillinen kameramoduuli. Neuroverkon kiihdyttämiseen työssä käytettiin teknologiayritys Intelin tuotteistamaa Intel® Myriad™ 2 Visual Processor -piirin sisältävää tietokoneeseen ulkoisesti liitettävää ”neuroverkkotikkua”, joka on kaupalliselta nimeltään ”Movidius™ Neural Computing Stick”. Laitteisto on kuvattu tarkemmin luvuissa 3 ja 4.

Työn tavoitteena on selvittää, miten hyvin kyseinen laite ja ohjelmistokokonaisuus soveltuu konenäön tuotekehitys- ja tutkimustyöhön. Tätä varten työssä kehitetään sovelluksia järjestelmän suorituskyvyn testaamiseen, sekä esimerkkitoteutukset reaaliaikaiseen kuvan luokitteluun ja kohteiden paikantamiseen kuvasta. Työssä tehtyjen ohjelmistojen keskeiset osat kuvataan luvussa 4, jossa myös esitellään käytetyt neuroverkkomallit. Luvussa 5 analysoidaan järjestelmän prosessointinopeutta, kuormitusta ja tehonkulutusta. Neuroverkkokiihdyttimen hyödyllisyyttä tutkitaan vertaamalla kiihdytettyä toteutetusta ohjelmistopohjaiseen toteutukseen. Luvussa 6 esitettään tutkimuksen yhteenveto.

2. NEUROVERKOT

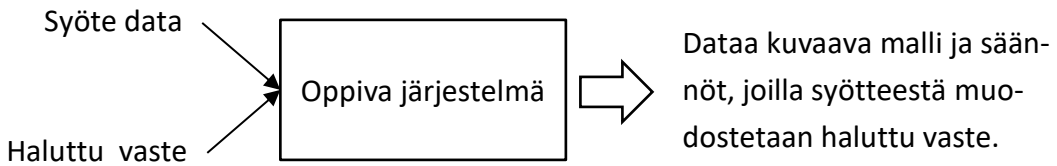
Tässä luvussa perehdytään yleisesti tekoälyyn, koneoppiseen, konenäköön sekä neuroverkkoympäristöihin, sekä täsmällisemmin tässä tutkimuksessa käytettyjen teknologioiden taustaan. Alueesta on saatavilla runsaasti lähdemateriaalia, tässä luvussa on yleisinä lähteinä käytetty pääasiallisesti julkaisuja ”Deep Learning” [4] sekä ”Neural Networks and Deep Learning” [17].

2.1 Koneoppiminen

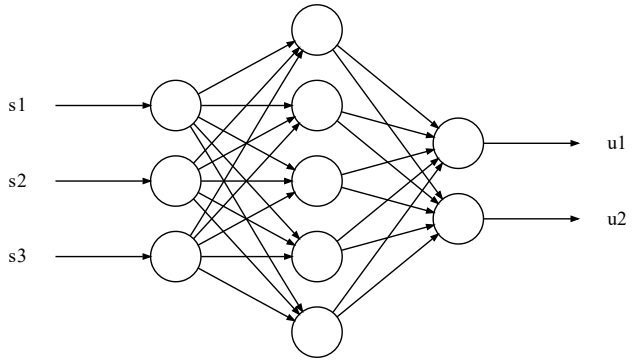
Tekoäly on yleisnimitys erilaisille tietojenkäsittelyjärjestelmille, joilla pyritään ratkomaan ongelmia, joissa tarvitaan ihmisaivojen kaltaista rakenteiden hahmottamista ja päättelyä. Sovellusalueita on lukuisia ja uusia kehitetään jatkuvasti. Tyypillisiä käyttökohteita tekoälymenetelmille ovat esimerkiksi esineiden löytäminen ja tunnistaminen kuvasta, puheen muuttaminen tekstiksi, kielen kääntäminen tai itsenäisesti ajava auto. Myös merkittäviä virstanpylväitä tekoälytutkimuksen historiassa ovat toteutukset, jotka kykenevät ylittämään ihmisen suorituskyvyn suosituissa peleissä kuten shakissa [12] ja go:ssa [2]. Viimeksi mainitut esimerkit ovat nostaneet tekoälyn myös suuren yleisön tietoisuuteen.

Koneoppimisella tarkoitetaan tekoälyn menetelmiä, jossa ongelman ratkaisuun tarvittavaa logiikkaa ei suoraan koodata vaan järjestelmä pyrkii analysoimaan kerättyä aineistoa ja muodostamaan tästä mallin, jota voidaan soveltaa uuteen aiemmin tuntemattomaan dataan.

Oppiminen voidaan jakaa kahteen eri pääluokkaan: ohjattuun ja ohjaamattomaan. Ohjatussa oppimisessa järjestelmälle osoitetaan syöte, esimerkiksi kuvia ja haluttu ulostulo kuten esimerkiksi tieto mitä kuvat esittävät. Järjestelmä pyrkii muuttamaan toimintaansa siten, että ero ulostulon ja halutun ulostulon välillä olisi mahdollisimman pieni. Ohjaamattomassa oppimisessa järjestelmä pyrkii etsimään syötedatasta säännönmukaisuuksia ja rakenteita ilman ulkoista palautetta. Kuvassa 1 esitetään ohjatun oppimisen periaate.



Kuva 1. *Ohjatun koneoppimisen periaate.*



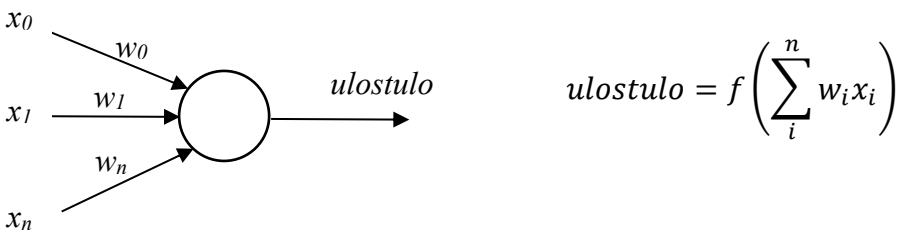
Kuva 2. *Yksinkertainen neuroverkko.*

2.2 Neuroverkkojen perusteet

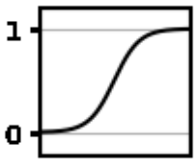
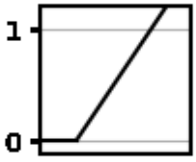
Neuroverkot ovat yksi yleisesti käytetty koneoppismalli. Ne ovat saaneet inspiraation aivojen tavasta käsitellä tietoa, jossa hermosoluista koostuva verkko käsittelee sähköisiä impulsseja. Vaikka neuroverkko voidaan toteuttaa myös elektronisena piirinä, joka käsittelee analogisia sähköisiä signaaleja samoin kuin biologinen esikuvansa, tyypillinen toteutus on kuitenkin laskennallinen algoritmi, joka suoritetaan digitaalisessa prosessorijärjestelmässä.

Neuroverkko opetetaan esimerkkidatan avulla tuottamaan syötteestä haluttu vaste. Tyypillinen sovellus on esimerkiksi hahmontunnistus, jossa verkon syötteenä on kuvadata ja ulostuloksi halutaan tieto siitä millä todennäköisyydellä tietty esine on kuvassa. Yleisesti neuroverkkoja voidaan käyttää hyvin laaja-alaisesti erityyppisiin informaatioprosessointitehtäviin.

Keinotekoinen neuroverkko voidaan ajatella suunnattuna graafina, jossa jokainen solmukohta tekee jonkun operaation syötteelle ja siirtää tuloksen eteenpäin seuraaville verkon solmupisteille eli neuroneille. Kuva 2 esittää yksinkertaista neuroverkkoa. Siinä syötedata (s_1, s_2, s_3) kulkee kolmen *neuronikerroksen* läpi ja verkon ulostulo (u_1, u_2) muodostuu aikaisempien kerrosten neuronien datalle tehdyistä operaatiosta. Neuroverkon rakenne, kerrosten määrä ja neuronien tyyppi ja se miten ne kytkeytyvät toisiinsa määräytyvät sen mukaan millaista datan prosessointia verkon halutaan tekevän.



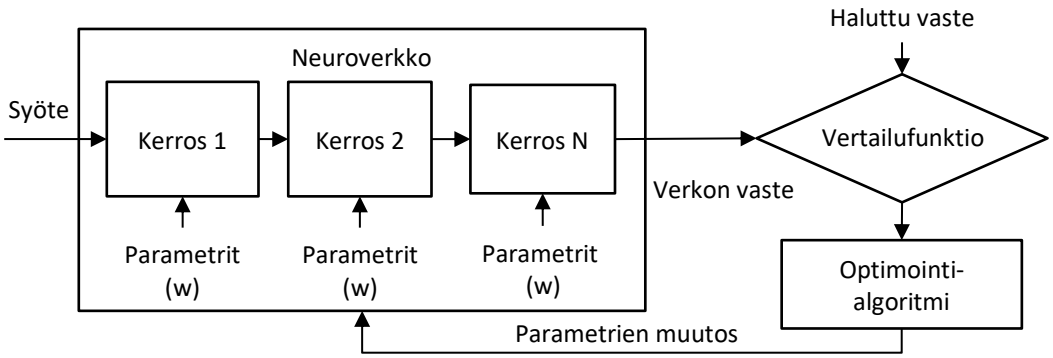
Kuva 3. *Esimerkki yksittäisestä neuronista.*

Neuronin tyyppi	Aktivaatiofunktio	Esimerkkikuvaaja
Logistinen Sigmoid neuroni	$f(\bar{x}) = \frac{1}{1 + e^{-\sum_i w_i x_i}}$	
Rajoitettu lineaarinen neuroni (RLU, engl. Rectified Linear Unit)	$f(\bar{x}) = \max\{0, \sum_i w_i x_i\}$	

Taulukko 1. *Esimerkkejä neuronien aktivaatiofunktiosta.*

Kuva 3 esittää verkon yksittäistä neuronia ja vastaavaa matemaattista operaatiota. Neuronin sisäänmeno on *n* kappaletta lukuarvoja $x_1, x_2, \ldots, x_n$. Jokainen sisäänmeno kerrotaan omalla painokertoimellaan w_i, lasketaan yhteen ja syntynyt summa on syöte funktiolle *f*. Funktiota *f* kutsutaan neuronin *aktivaatiofunktioksi.* Verkon eri neuroneilla voi olla erilaisia aktivaatiofunktiota. Taulukossa 1 on esitetty kaksi yleisesti käytettyä vaihtoehtoja. Aktivaatiofunktion valinta vaikuttaa ratkaisevasti neuronin toimintaan.

Tyypillisesti neuroverkon opettaminen tapahtuu muuttamalla neuronien aktivaatiofunktioiden painokertoimia iteratiivisesti optimointialgoritmeilla, jotka pyrkivät minimoimaan eron saavutetun ja halutun ulostulon välillä. Tätä varten tarvitaan opetusdataa, jossa jokaista syötettä kohden on olemassa ennalta määritelty haluttu vaste. Opettaminen tapahtuu antamalla verkolle syöte ja laskemalla verkon vaste. Ulostuloa verrataan haluttuun tulokseen sopivan *kustannusfunktion* avulla. Jokaisen opetussyötteen jälkeen verkon painokertoimia muutetaan optimointialgoritmilla, kunnes saavutetaan tila, jolloin kustannusfunktion arvo ei enää pienene. Kuva 4 esittää edellä kuvatun neuroverkon opettamisen periaatteen. Yleisesti käytetyt käytännön optimointimenetelmät pohjautuvat koko verkkoa kuvaavan kustannusfunktion minimin löytämiseen stokastisella gradientti pohjaisella haulla. Keskeinen työkalu tässä menetelmässä on ns. *takaisinlevitysalgoritmi,* (engl. *backprogation*), jonka avulla voidaan tehokkaasti päivittää verkon painokertoimia [21].



Kuva 4. *Neuroverkon opettaminen.*

2.3 Syvät neuroverkot ja konvoluutioneuroverkot

Syvät neuroverkot (*Deep Neural Networks*, DNN) ovat verkkorakenteita, jotka muodostuivat lukuisista kerroksista. Kun yksinkertaisessa verkossa on esimerkiksi vain kolme kerrosta: sisäänmeno, yksi piilokerros ja ulostulo voi syvässä verkossa olla kymmeniä tai satoja piilokerroksia. DNN voidaan ajatella järjestelmänä, jossa jokainen kerros luo uuden esityksen sisäänmenodatasta, ja jokainen kerroksen jälkeen prosessoitu data on lähempänä halutun ulostulon esitysmuotoa.

Konvoluutioneuroverkot (*Convolutional Neural Networks* CNN) sisältävät kerroksia, jotka toteuttavat konvoluutio-operaatiota. Nämä verkot ovat tulleet suosituksi erityisesti kuvankäsittelyssä. Tyypillisesti verkon ensimmäiset kerrokset muodostuvat konvoluutio-operaatiosta, jotka pyrkivät tunnistamaan kuvasta erilaisia yksinkertaisia rakenteita kuten kohteiden reunoja. Verkon myöhemmät kerrokset tunnistavat useamman rakenteen olemassaoloa ja niiden keskinäistä sijaintia ja muodostavat esitystä monimutkaisempien piirtein olemassaolosta kuvassa. Konvoluutiokerroksien jälkeen verkossa on tyypillisesti epälineaarisilla aktivaatiofunktiolla varustetuista neuroneista muodostuneita täysin kytkeytyneitä kerroksia, jotka oppivat dataan liittyviä monimutkaisempi suhteita.

2.4 Neuroverkkojen ohjelmistoympäristöt

Neuroverkot voidaan ajatella joukkona peräkkäistä vektori- tai laajemmin tensorioperaationa. Tämä mahdollistaa tehokkaan totutuksen laiteympäristöissä, joissa on rinnakkaiseen laskentaan soveltuvia prosessoreita, mutta tarjoaa myös hyödyllisen tavan esittää neuroverkon laskennan laiteriippumattomalla tavalla.

Syvien neuroverkkojen toteutukseen onkin tarjolla useista ohjelmistoympäristöjä, jotka helpottavat ohjelmointia sekä mahdollistavat laskennan optimaalisen toteutetukseen erilaisissa laitteistoissa. Näiden avulla verkkoja voidaan rakentaa parametrisoitavista osista kuten esimerkiksi konvoluutio-, yhdistely ja täysin kytketyistä epälineaarisista kerroksista. Ohjelmointiympäristöt sisältävät näiden eri kerrosten optimoituja toteutuksia erilaisille alustoille ja helpottavat verkon tehokasta toteutusta. Tyypillisesti verkon suunnittelijan ei tarvitse ymmärtää tietyn laitteiston yksityiskohtia ja verkon siirtäminen erilaiselle alustalle on monesti suoraviivaista. Seuraavassa lyhyt katsaus keskeisiin ympäristöihin:

TensorFlow™ [22] on alun perin Googlen kehittämä avoimen lähdekoodin ohjelmakirjasto, joka on tarkoitettu tehokkaaseen numeeriseen laskentaan. Sen joustava arkkitehtuuri mahdollistaa laskennan toteutuksen erilaisilla alustoilla kuten esimerkiksi yleiskäyttöisillä prosessoreille (Central Processing Unit, CPU) tai grafiikkakiihdyttimillä (Graphics Processing Unit, GPU).

Keras [13] on korkean tason ohjelmointirajapinta Python ohjelmointikielellä neuroverkkojen toteuttamiseen. Keras ei toteuta varsinaista laskentaa itse vaan käyttää tehokasta

tensorilaskentaa tukevia ohjelmakirjastoja kuten TensorFlow™, Microsft Cognitive Tookit (CNTK) [15] tai Theano [23].

Caffe [1] on Berkleyn yliopistossa kehitetty ohjelmistoympäristö syväoppimiseen. Samoin kuin TensorFlow™ se tarjoaa korkean tason rajapinnan neuroverkkojen kehittämiseen, sekä mahdollisuuden toteuttaa laskenta joko CPU:lla tai GPU:lla.

OpenVino™ [8] on Intelin kehittämä avoimen lähdekoodin ohjelmistokirjasto konvoluutioneuroverkkojen tehokkaaseen toteuttamiseen Intelin tukemilla arkkitehtuureilla [5]. Keskeisenä ajatuksena on esittää verkot laitteistoriippumattomalla kuvauksella ja ohjelmointirajapinnalla, jota voidaan käyttää optimoituna eri toteutusympäristöissä.

OpenCV [18] on yleiskäyttöinen kuvankäsittelyyn ja konenäköön tarkoitettu avoimen lähdekoodin ohjelmakirjasto. Osana kirjastoa on syvien neuroverkkojen toteutusta varten tehtyä moduuleja.

2.5 Valmiit neuroverkkorakenteet

Tutkijat ja yritykset ovat kehittäneet useita neuroverkkorakenteita, jotka soveltuvat erilaisten ongelmien ratkaisuun. Yleisesti käytetyt ohjelmointiympäristöt mahdollistavat näiden valmiiden verkkojen käyttämisen erilaisissa sovelluksia suhteellisen suoraviivaisesti ja tällöin sovelluskehittäjän ei välttämättä tarvitse tietää yksityiskohtia verkon rakenteesta tai sen opettamiseen käytetystä datasta.

Haasteena tosin on, että käytetyt ympäristöt eivät tue kaikki täsmälleen samankaltaisia operaatiota ja ympäristöstä toiseen siirtyminen ei aina ole ongelmatonta. Toisaalta valmis verkko ei myöskään aina ole optimaalinen erilaisessa ympäristössä, esimerkiksi konenäkösovelluksen suorituskyky voi vaihdella käytetyn kameran laadun mukaan ja hyvälaatuisella kameralla hyvissä olosuhteissa opetettu neuroverkko ei välttämättä tuota samaa tulosta haastavammissa olosuhteissa toimivalla kameralla.

On myös mahdollista käyttää jo hyväksi todettuja verkkojen topologioita uusissa sovellusalueissa. Kehitetyt verkkotopologiat voivat olla optimoituja tiettyyn sovellukseen ja toteutuksen kompleksisuuteen. Käyttämällä tunnettua rakennetta mutta opettamalla verkko uuden sovelluksen tarvitsemalla datalla kehittäjä pystyy saamaan aikaseksi verkon, joka toimii tietyssä ympäristössä.

3. NEUROVERKKOJEN KIIHDYTTÄMINEN

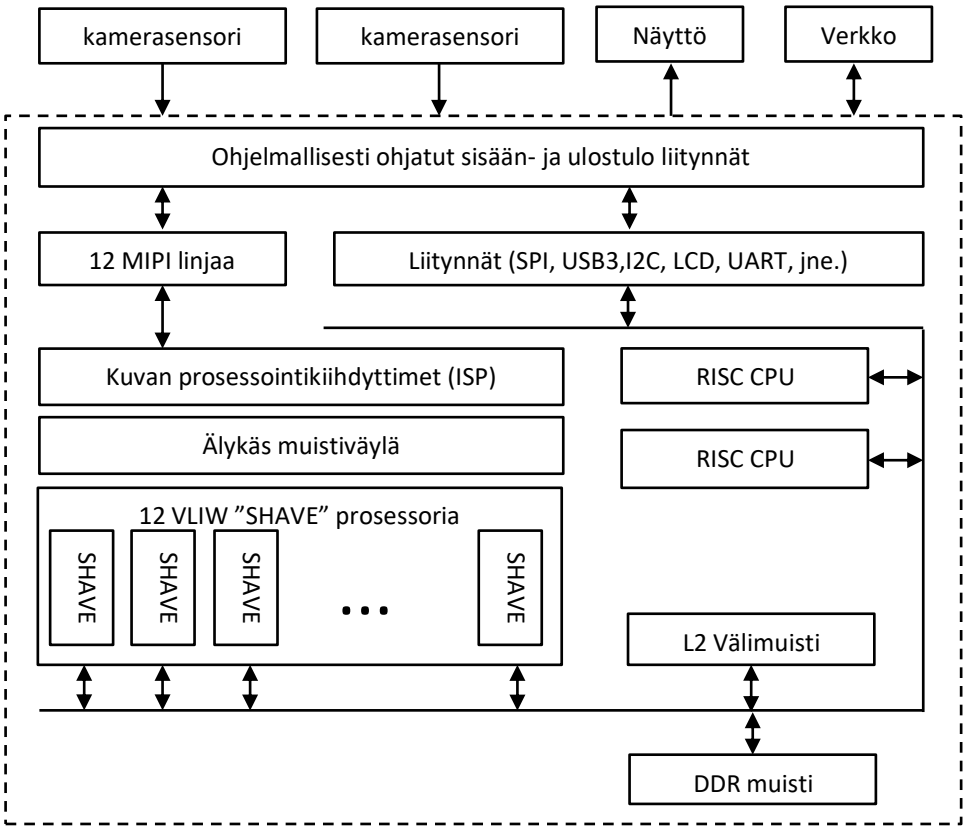
Neuroverkot soveltuvat toteuttaviksi hyvin ympäristöissä, jotka tukevat tehokasta matalan tason rinnakkaista prosessointia. Tyypillisessä neuroverkon tensorioperaatiossa tehdään sama laskentaoperaatio useille syötteille ja näin ollen se soveltuu hyvin SIMD (Single Instruction Multiple Data) tyyppisille prosessoriarkkitehtuureille. Samankaltaista prosessointia käytetään myös useissa kuvankäsittely- ja tietokonegrafiikkasovelluksissa. Näin ollen erilaiset vektoriprosessointiarkkitehtuurit, kuten esimerkiksi grafiikkakiihdyttimet vektoriyksiköt ja keskusyksikköjen SIMD käskykantalaajennukset tukevat hyvin myös neuroverkkojen kiihdytystä. Suositut neuroverkko-ohjelmointiympäristöt tarjoavatkin hyvin tukea yleisille grafiikkakiihdyttimille ja CPU käskykannoille.

Pienemmän tehonkulutuksen laitteissa ja sovelluksissa neuroverkkojen kiihdytykseen on kuitenkin toistaiseksi tarjolla vähemmän vaihtoehtoja. Esimerkiksi useimpien kannettaviin laitteisiin suunniteltujen piirisarjojen grafiikkaprosessoreiden suorituskyky ei ole samalla tasolla kuin PC ympäristössä. Nähtävissä on kuitenkin, että erityisesti syviä neuroverkkoja käyttävien tekoälysovellusten määrän lisääntyessä piirivalmistajat tulevat lisäämään erityisiä kiihdytinratkaisuja myös sulautettuihin järjestelmiin suunnattuihin piireihin. Seuraavassa kappaleessa esitellään yksi neuroverkkokiihdytykseen sulautetussa järjestelmässä soveltuva prosessori.

3.1 Myriad 2 arkkitehtuuri

Myriad 2 MA2x5x Vision Processor Unit (VPU) [9] on Intelin luoma integroitu systeemipiiri, joka on tarkoitettu visuaalisen informaation käsittelyyn erityisesti pientä tehonkulutusta vaativissa sulautetuissa sovelluksissa. Piirin nimellinen tehonkulutus on 1 W ja se soveltuu näin ollen hyvin käytettäväksi esimerkiksi kameravalvontaan, miehittämättömiin ilma-aluksiin, robotiikkaan ja puettavaan elektroniikkaan. Tyypillisiä sovelluksia ovat konenäkö, erityisesti neuroverkkoihin perustuva luokittelu, syvyys ja etäisyystiedon laskeminen, eleohjaus ja visuaaliseen informaatioon pohjautuva navigointi.

Myriad 2 VPU tarjoaa itsenäisen kokonaisuuden, johon voidaan suoraan liittää MIPI-liitäntästandardin mukaisia kamerasensoreita, prosessoida kameralta tulevaa dataa sisäisellä kuvankäsittelyprosessorilla (ISP) ja toteuttaa erilaisia kuvankäsittelyn ja hahmontunnistuksen algoritmeja siihen optimoidulla prosessoreilla. Piirissä on 12 *pitkän sanapituuden* (VLIW) arkkitehtuuriin perustuvaa SHAVE-vektoriprosessoria. Nämä prosessorit ovat suunniteltu tehokkaaseen rinnakkaiseen laskentaan ja soveltuvat hyvin algoritmeihin, joissa suuri määrä dataa käsitellään samankaltaisilla operaatioilla. Piirin suunnitellussa on myös huomioitu energiatehokas datan siirto eri prosessoreiden välillä, esimerkiksi ISP kommunikoi SHAVE yksiköiden kanssa ilman datan siirtoa keskusmuistiin.



Kuva 5. *Myriad 2 MA2x5x lohkokaavio[9].*

Tämä mahdollistaa pienemmän tehonkulutuksen, sekä nopeamman prosessoinnin verrattuna järjestelmään, jossa data kulkisi ulkoisen muistin kautta. Kuvassa 5 esitetään ylätason lohkokaavio Myriad 2 pohjaisesta järjestelmästä.

Piirin valmistaja tarjoaa valmista kehitysympäristöä *”Myriad™ Development Kit & Board”*, joka mahdollistaa nopean sovelluskehityksen erityisesti tuotekehityksen alkuvaiheessa. Tämä on hyvä lähtökohta, kun tehdään tuotetta, joka hyödyntää Myriad 2 järjestelmää kokomaisuudessaan. Jos kuitenkin tavoitteena on hyödyntää lähinnä Myriad 2 piirin rinnakaislaskenta ominaisuuksia tutkimus- ja varhaisen kehitysvaiheen työssä on tarjolla muitakin vaihtoehtoja. Myriad 2 VPU piiristä on saatavilla Movidius Neural Computing Stick (MNCS) [10], koteloitu USB porttiin liitettävä laite, joka soveltuu hyvin tutkimukseen, kokeiluihin ja jopa pienisarjaisiin kaupallisiin tuotteisiin. Nimensä mukaisesti Myriad 2 piiriä käytetään tässä laitteessa vain kiihdyttämään neuroverkkolaskentaa. Varsinainen sovellus ja mahdolliset liitynnät ulkomaailmaan on hoidettavat sen järjestelmän kautta mihin MNCS liitetään USB väylän avulla. Laite esitellään tarkemmin luvussa 3.2.

Myriad 2 piiri on myös helposti saatavilla Googlen tuottaman *”AIY Vision Kitin”* osana[5]. Tässä kokonaisuudessa on mukana Raspberry Pi Zero pienoistietokone, kamera ja yhteensopiva piirilevy, joka sisältää Movidius Myriad 2 VPU prosessorin.

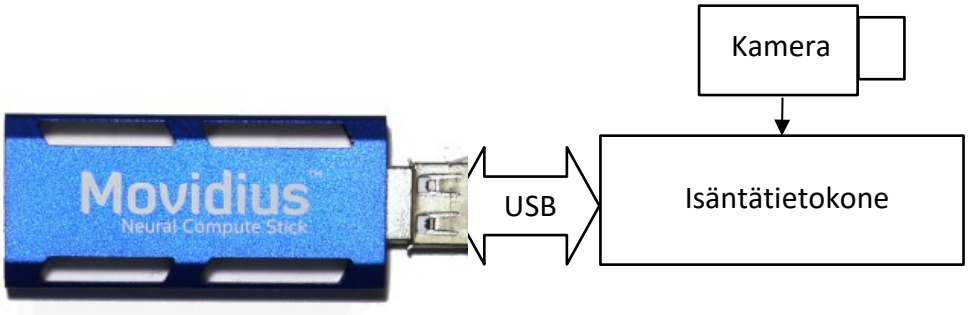
3.2 Neurolaskentakiihdytin USB-väylään

”Neuroverkkotikku”, Movidius™ Neural Computing Stick (MNCS) tarjoaa mahdollisuuden kiihdyttää neuroverkkoa tietokonejärjestelmässä, johon se voidaan kytkeä USB väylän avulla (Kuva 6). Laitteessa ei ole muita liitäntöjä ja isäntäjärjestelmänä toimiva tietokone vastaa datan siirrosta esimerkiksi kameralta neuroverkkokiihdyttimelle. Laite tukee 5 Gbit/s tai 480 Mbit/s tiedonsiirtonopeuksia isäntäjärjestelmän kyvykkyyden mukaan. Virransyöttö tapahtuu myös USB:n kautta.

MNSC sisäisenä prosessorina toimii edellä kuvattu Myriad 2 piiri. Laitteen toiminta on suoraviivaista. Kun se kytketään isäntäjärjestelmään USB liitynnän kautta se käynnistyy ja ensin sisäisen laiteohjelmiston avulla ja näyttäytyy USB 2.0 standardin mukaisena laitteena. Isäntälaitteen ohjelmisto kommunikoi MNSC kanssa neurolaskenta ohjelmointirajapinnan (NCAPI) kautta. Kun laite alustetaan tämän rajapinna komennoilla, sinne latautuu uusi laiteohjelmisto, joka tukee neurolaskenta kehitysympäristöä (MNSC SDK). Tässä vaiheessa laite käynnistyy uudelleen ja liittyy isäntäjärjestelmään uutena USB laitteena, joko USB2.0 tai USB3.0 standardin mukaisesti. Nyt laite on valmiina ottamaan vastaan neuroverkkoa määritteleviä graafitiedostoja ja toteuttamaan laskentaa.

Kun laitteella halutaan toteuttaa neurolaskenta, ladataan ensin neuroverkon määrittävä graafitiedosto. Myriad piirin RISC prosessori alustaa SHAVE vektoriprosessorit toimimaan määritellyn neuroverkon mukaisesti. Tämän jälkeen voidaan verkolle syöttää sisäänmenodata ja lukea verkon tulos USB väylän kautta.

Neuroverkko, jota halutaan käyttää järjestelmässä, tulee kääntää sopivaksi graafitiedostoksi. MNSC SDK tarjoaa tähän työkaluja, joista tärkein on **mvNCCompile**, jolla voidaan muuntaa Caffe tai TensorFlow ympäristöllä tehty neuroverkko kiihdyttimen ohjelmointirajapinnan käyttämään muotoon. Testiohjelmaa **mvNCCheck** voidaan käyttää varmistamaan, että verkko toimii suunnitellusti. Ohjelma suorittaa neuroverkon laskennan sekä neuroverkkotikussa, että isäntälaitteessa ohjelmallisesti ja vertaa tuloksia. Kolmas hyödyllinen työkaluohjelma on **mvNCProfile,** joka tuottaa raportin verkon eri kerrosten suoritusnopeudesta ja muistin käytöstä. Työkalu on käyttökelpoinen, kun optimoidaan verkkoa juuri tätä kiihdytysympäristöä varten.



Kuva 6. *Movidius neuroverkkokiihdytin USB liitäntään.*

Ohjelmointiympäristö (MNCS SDK) on vapaasti ladattavista GitHub-sivustosta [10]. Tarjolla on ohjelmointi rajapinnat C++ ja Python ohjelmointikieliin. Ympäristö tarjoaa mahdollisuuden käyttää suoraan Caffe tai TensorFlow kirjastoilla tehtyjä malleja, tosin kaikki ympäristöjen tarjoamat ominaisuudet eivät ole tuettuja ja näin ollen kaikkien valmiiden verkkojen käyttäminen ei ole aina ole suoraviivaista. On myös huomiotava, että ohjelmointiympäristö ei tue verkon opettamista ja ennen mallin muuntamista kiihdyttimelle sopivaksi siitä tulee poistaa opetuksessa käytetyt osat.

MNSC SDK tukee virallisesti PC ympäristössä Linux käyttöjärjestelmän laitteita ja myös Raspberry Pi tietokonetta. Vaihtoehtoisena ohjelmointirajapinta MNSC laitteille on myös OpenVino kirjasto. Valitettavasti OpenVino ei kirjoittamishetkellä tue virallisesti neuroverkkotikkua muussa kuin PC ympäristössä, joten sen käyttö pienoistietokoneen kuten Raspberry Pi:n kanssa ei ole vielä mahdollista.

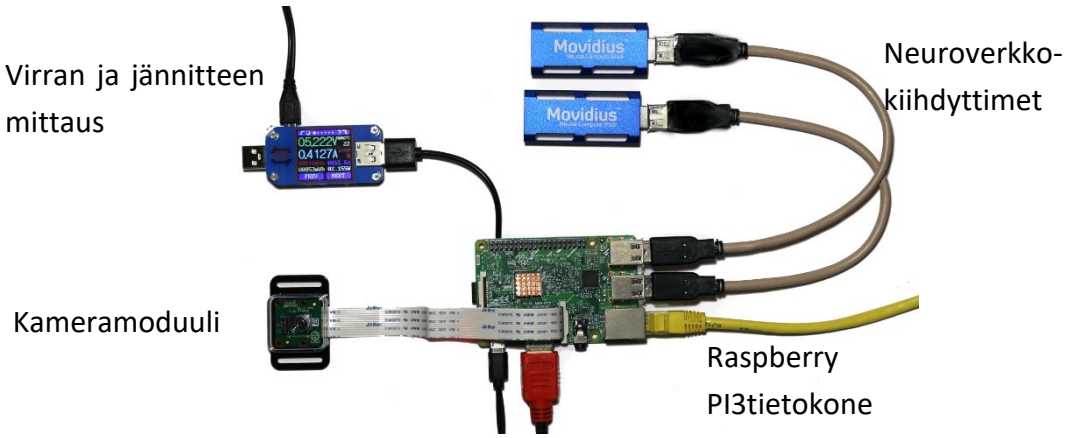
4. KONENÄKÖSOVELLUS RASPBERRY PI-YMPÄRISTÖSSÄ

Työssä tutkittiin Raspberry Pi pienoistietokoneen ja Movidius™ Neural Computing Stick laitteen soveltuvuutta neuroverkkoja käyttävän konenäkösovelluksen toteuttamiseen. Tavoitteena oli selvittää minkälaiseen suorituskykykyyn järjestelmällä päästään ja miten sujuvaa erilaisten ratkaisujen tekeminen on. Tätä varten kehitettiin testiohjelmistoja erilaisten toteutusten vertailuun, sekä Raspberry Pi:n kameramoduulin kanssa toimiva reaaliaikainen konenäkösovellus. Sovellukseen valittiin kaksi tyypillistä konenäköongelmaa, kuvan luokittelu ja tiettyjen kohteiden paikantaminen kuvasta.

4.1 Laiteympäristö

Tutkimuksessa käytetty laiteympäristö koostui Raspberry Pi 3 Model B pienoistietokoneesta, kahdesta Movidius™ Neural Computing Stick kiihdyttimestä ja 8 megapikselin Raspberry Pi v2 kameramoduulista [20]. Joissain testeissä käytettiin myös erillistä omalla virtalähteellä varustettua USB-keskitintä, johon neuroverkkokiihdyttimet kytkettiin. Tämän lisäksi käytössä oli virrankulutuksen ja jännitteen mittauslaitteita. Laitteet ovat esitelty kuvassa 7.

Raspberry Pi 3B tietokoneen keskusyksikkönä toimii Broadcomin valmistamaa BCM2837 järjestelmäpiiri, jossa on 4 ARMv8 prosessoriydintä [20]. Prosessorin kellotaajuus on 1200 MHz. Piirissä on myös erillinen grafiikan, videon ja kameradatan käsittelyyn tarkoitettu 400 MHz ”VideoCore IV” -yksikkö. Käytetyssä laitteistossa asennettiin järjestelmäpiirin päälle kuparinen jäähdytyssiili, joka auttoi lämpötilan rajoittamisessa suurilla prosessorikuormilla.



Kuva 7. *Tutkimuksessa käytettyä laitteistoa.*

Pienoistietokoneen piirilevyllä on 4 ulkoista USB porttia, verkkoliitäntä, HDMI videoliitäntä, sekä yleiskäyttöisiä liitäntöjä erilaisille lisälaitteille. Laite tarjoaa myös langattomat ”Bluetooth”- ja lähiverkkoliitynnän. Järjestelmän tehonsyöttö tapahtuu 5 V mikro-USB liitynnän kautta, suositeltu virtalähteen virranantokyky on 2.5 A. Laite kykenee tuottamaan USB liitäntöihin maksimissaan yhteensä 1,2 A virran, joka tässä tutkimuksessa osoittautui jossain tilanteissa riittämättömäksi useammalle neuroverkkokiihdyttimelle. Näissä tilanteissa kiihdyttimet kytkettiin erilliseen USB keskittimeen. Raspberry Pi tukee USB 2.0 standardia, jonka maksimi tiedonsiirtonopeus on 480 Mbit/s. On kuitenkin huomioitava, että kaikki ulkoiset portit ja verkkoliityntä jakavat tämän siirtokaistan keskenään.

Käytetyn kameramoduulin sensorina on Sonyn valmistama IMX219, jonka resoluution on 3280x2464 kuvapikseliä. Videon kuvanopeus 1240x1080 pikselin resoluutiolla on 30 kuvaa sekunnissa ja 640x480 pikselin resoluutiolla voidaan saavuttaa 90 kuvan sekuntinopeus. Kamerassa on suhteellisen laajakulmainen optiikka, joka tarjoaa 62,2 asteen vaakasuuntaisen ja 48,8 asteen näkymän.

4.2 Ohjelmointiympäristö

Raspberry Pi:n käyttöjärjestelmänä käytettiin ”Raspbian Stretch” nimistä Debian Linux pohjaista järjestelmää. Tähän ympäristöön on saatavilla lukuisia ohjelmistoja ja ohjelmakirjastoja ja niiden asentaminen on suoraviivaista käyttöjärjestelmän tarjoamilla työkaluilla. Työssä kehitettiin sovelluksia Python ohjelmointikielellä ja tekemisessä hyödynnettiin yleisesti saatavilla ohjelmakirjastoja. Näistä sovelluksen kannalta keskeisimmät on listattu taulukossa 2.

Ohjelmisto	Versio	Kuvaus	Saatavilla
python	3.5.3	Ohjelmointikieli	https://www.python.org/
picamera	1.13	Raspberry Pi kameran rajapinta	https://picamera.readthedocs.io
mncs	2.05.00.02	Neuroverkkotikun ohjelmointirajapinta MNCS SDK	https://movidius.github.io/ncsdk/index.html
cv2	3.3.0	OpenCV kuvankäsittely- ja konenäkökirjasto	https://opencv.org/
numpy	1.15.1	Numeerisen laskennan kirjasto	http://www.numpy.org/

Taulukko 2. *Käytetyt keskeiset ohjelmistot.*

Picamera-kirjasto tarjoaa helppokäyttöisen liittymän järjestelmän kameratoiminnallisuuteen ja soveltuu hyvin kuvaa analysoivan konenäkösovelluksen toteuttamiseen. Eräs ohjelmointimalli reaaliaikaiselle kuva-analyysille tätä kirjastoa käyttäen on esitetty ohjelmakoodi esimerkissä 1. Siinä luodaan ensi uusi kuva-analyysiä tekevä luokka periyttämällä Picamera-kirjaston tarjoaman analyysiluokka ja uudelleenmäärittelemällä sen *analyze()*-metodi, jossa varsinainen prosessointi tapahtuu. Seuraavaksi alustetaan kamera haluttuun tarkkuuteen ja kuvanopeuteen. Lopuksi käynnistetään kuvanotto, joka ryhtyy syöttämään kuvia aiemmin määritellylle analyysimetodille.

```
# Tämän luokan analyze() metodia kutsutaan jokaisella kuvalla
# prosess() funktio tekee varsinaisen kuva-analyysin
class Analyzer(picamera.array.PiRGBAnalysis):
    def analyze(self, image):
        process(image)

# Avataan kamera haluttuun resoluutioon ja kuvataajuuteen
with picamera.PiCamera() as camera:
    camera.resolution = (227,227)
    camera.framerate = 15

# Käynnistetään kuvanotto 60s ajaksi ja syötetään kuvat
# detector-olion analyze metodille.
    with Analyzer(camera) as analyzer:
        camera.start_recording(analyzer, format='rgb')
        camera.wait_recording(60)
        camera.stop_recording()
```

Ohjelma 1. *Esimerkki Picamera kirjaston käytöstä.*

MNCS SDK tarjoaa myös suoraviivaisen ohjelmointirajapinnan neuroverkon käyttöön. Ohjelmalistaus 2 sisältää esimerkin periaatteesta. Koodin alussa etsitään ja alustetaan laitteistoon kytketty neuroverkkokiihdytin. Seuraavaksi laitteeseen ladataan *mvNCCompile*-kääntäjällä etukäteen tehty verkkograafi. Lopussa luodaan funktio, joka syöttää annetun kuvan kiihdyttimelle ja hakee laskennan tuloksen.

```
# Etsitään laitteistoon liitetty ensimmäinen kiihdytin.
device = mvnc.enumerate_devices()[0]
device.open()

# Ladataan verkkoa kuvaava käännetty graafi kiihdyttimeen
graph = mvnc.Graph('graph')
fifoIn, fifoOut = graph.allocate_with_fifos(device, compiled_graph)

# Funktio, joka analysoi annetun kuvan neuroverkolla.
def process(image):
    # Lähetetään kuva kiihdyttimen neuroverkolle
    graph.queue_inference_with_fifo_elem(fifoIn, fifoOut, image, 0)
    # Luetaan neuroverkon laskennan tulos
    result, userobj = fifoOut.read_elem()
```

Ohjelma 2. *Esimerkki MNCS ohjelmointirajapinnan käytöstä.*

Myös **OpenCV** kirjaston DNN-moduuli on vastaavalla tavalla helppokäyttöinen. Ohjelmalistaus 3 antaa esimerkin sen käytöstä. Aluksi verkkoa kuvaava malli ladataan tiedostosta. Prosessointifunktiossa kuva muutetaan verkolle sopivaan muotoon, asetetaan sisäänmenoksi ja lopuksi suoritetaan laskenta ja saadaan tulos. On huomattava, että OpenCV kirjasto osaa lukea suoraan useita yleisesti käytettyjä neuroverkkoformaatteja.

```
# Luodaan verkko tiedosta ladatusta mallista.
net = cv2.dnn.readNetFromCaffe('prototxt', 'alexnet.caffemodel' )

# Funktio, joka analysoi annetun kuvan neuroverkolla.
def process(image):
    # Muunnetaan kuva neuroverkolle sopivaan muotoon
    blob = cv2.dnn.blobFromImage(image,1,(227,227))
    # Asetetaan kuva neuroverkon sisäänmenoksi
    net.setInput(blob)
    # Suoritetaan verkon laskenta
    result = net.forward()
```

Ohjelma 3. *Esimerkki OpenCV DNN ohjelmointirajapinnan käytöstä.*

4.3 Testisovellukset

Työn ensimmäisessä vaiheessa kehitettiin testisovelluksia, joilla tutkittiin järjestelmän toimintaa ja suorituskykyä. Näissä toteutuksissa pyrittiin pitämään keskusyksikön ylimääräinen kuormitus pienenä, jotta nähtäisiin neuroverkkoprosessointiin kuluva aika ja sen vaikutus järjestelmän kokonaiskuormitukseen. Testiohjelmissa ei käytetty kameraa, vaan verkolle syötettiin etukäteen muistiin ladattua kuvaa. Testiajon aikana sovellus tallensi jokaisen kuvan käsittelyyn käytetyn ajan, jonka perusteella laskettiin sekä hetkellinen, että keskimääräinen kuvanopeus. Erillinen sovellus mittasi sekunnin välein verkon laskemiseen kuluvaa aikaa, sekä prosessorin lämpötilaa ja kellotaajuutta. Mittaussovelluksen vaikutus kokonaiskuormitukseen oli alle yhden prosenttiyksikön.

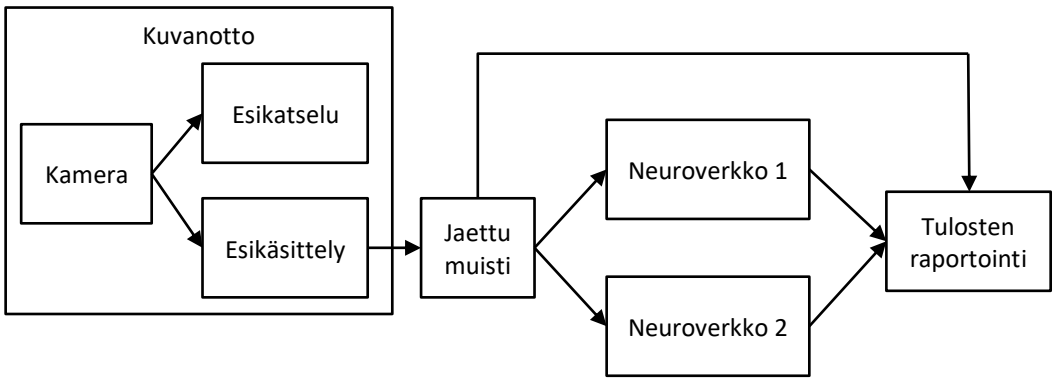
Ensimmäisessä vaiheessa testisovelluksesta tehtiin suoraviivainen yhdessä säikeessä toimiva versio, jolla testattiin vertailukohtana käytettyä ohjelmistopohjaista OpenCV kirjastolla suoritettavaa neuroverkon laskentaa. Tämä kirjasto on tehokkaasti optimoitu ja python rajapinta kutsuu C++ kielellä toteutettua ohjelmistoa, joka pystyy hyödyntämään kaikki Raspberry Pi:n prosessoriytimet. Joten vaikka Python koodi itsessään toimii vain yhden prosessoriytimen sisällä, ohjelmistopohjainen neuroverkko käyttää hyväkseen tietokoneen kaikkia 4 prosessoriydintä.

Vastaava yksisäikeinen toteutus tehtiin myös yhden neuroverkkokiihdyttimen käyttöön. Ohjelman rakenne oli samanlainen kuin aiemmassa testiohjelmassa, vain verkon alustus ja laskentaa toteuttava funktio oli erilainen, samaan tapaan kuin aiemmissa ohjelmalistauksissa 2 ja 3.

Useamman kiihdyttimen käyttöä varten toteutettiin rinnakkaista prosessointia käyttävä sovellus, jolla kiihdyttimiä voitiin käyttää samanaikaisesti. Python ohjelmointikieli tarjoaa kaksi tapaa rinnakkaisen prosessoinnin toteutukseen: *säikeistyksen* (engl. *multithreading*) ja *erillisten prosessien* (engl. *multiprocessing*) käytön. Näistä säikeistys on ohjelmoijan kannalta yksinkertaisempi tekniikka, koska rinnakkaiset säikeet jakavat saman sovelluskontekstin ja datan siirto eri säikeitten välillä on suoraviivaista, kunhan tarvittava luku ja kirjoitus operaatioiden synkronointi toteutetaan oikein. Prosessien välinen kommunikaatio on taas rajoittuneempaa ja vaatii siihen erityisesti suunniteltujen tietorakenteiden käyttöä.

Säikeistyksen teoreettisena haittapuolena on CPython tulkin toteutuksen yksityiskohta, niin sanottu ”global interpreter lock” (GLI) joka rajoittaa python tulkin toimimaan vain yhdessä prosessoriytimessä, jossa myös kaikki sovelluksen säikeet suoritetaan. Tämä tarkoittaa, että python koodi ei pysty hyödyntämään moniydin prosessoria mahdollisimman tehokkaasta. Sovelluksesta kuitenkin riippuu, onko tämä rajoitus merkittävä vai ei. Esimerkiksi jos python sovellus käyttää pääosaan prosessointia ohjelmakirjastoja, jotka on toteutettu C tai C++ kielellä, voidaan GLI rajoitus ohittaa. Multiprosessointi toteutuksessa ei edellä mainittua mahdollista suorituskykyrajoitusta ole. Käyttöjärjestelmä voi ajaa erillisiä prosesseja vapaasti kaikille ytimillä.

Datan siirto on kuvankäsittelysovelluksissa yleensä operaatio, jonka tehokkuuteen tulee kiinnittää huomiota, tarpeetonta kopiointia ja datan koodausta ohjelman osien välisiin siirtoprotokolliin tulisi välttää. Tehdyissä ohjelmistoissa ongelma ratkaistiin käytättämällä jaettua muistialuetta, joka toteutettiin siten, että se oli käytettävissä sekä erillisistä säikeistä tai prosesseista. Tässä tutkimuksessa rinnakkaista prosessointia käyttävät sovellukset toteutettiinkin käyttäen sekä säikeitä, että erillisiä prosesseja. Näin voitiin vertailla, onko toteutustavoilla vaikutusta suorituskykyyn. Ohjelmointiteknisesti toteutukset ovat hyvin samankaltaisia, koska python tarjoaa *multiprocessing*- ja *threading*-kirjastoissa lähes identtiset rajapinnat rinnakkaisten prosessinen tai säikeiden luomiseen.



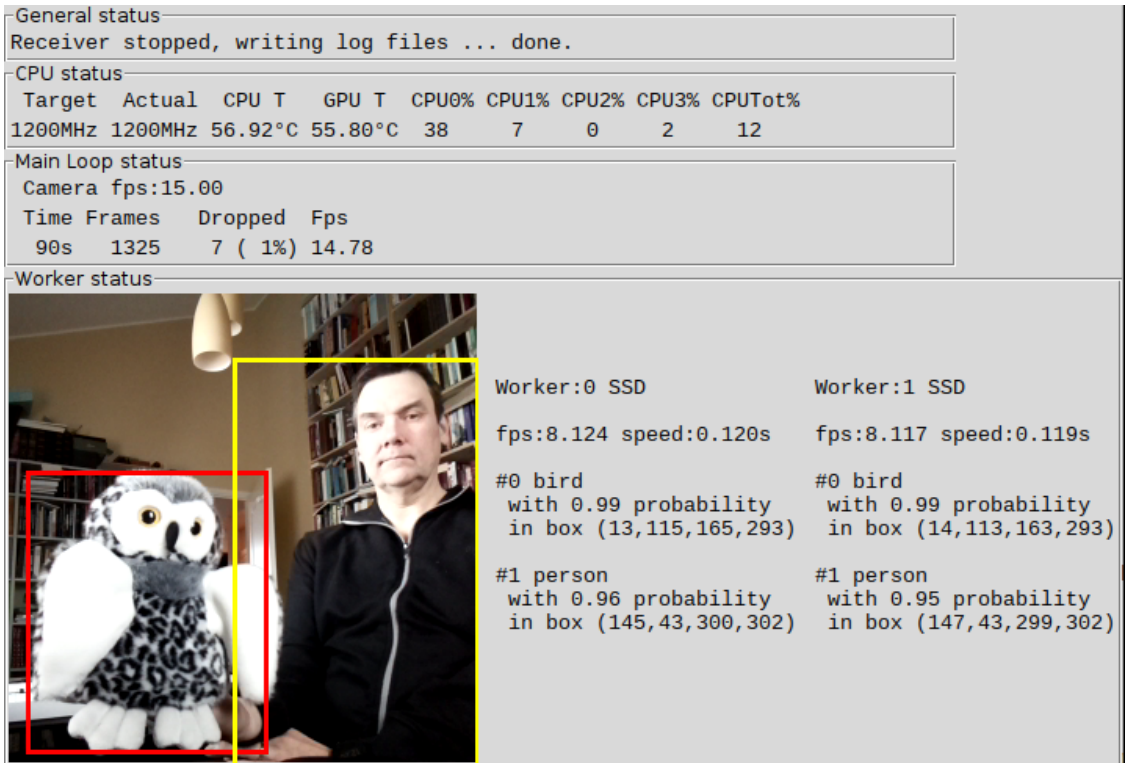
Kuva 8. *Rinnakkais-prosessointi ohjelman rakenne.*

4.4 Konenäkösovellus

Työn toisessa vaiheessa kehitettiin varsinainen konenäkösovellus, jossa Rasperry tietokoneeseen kytkettiin kamera, joka otti kuvia ja lähetti ne edelleen neuroverkon prosessoitavaksi. Kameran toteutuksessa käytettiin *picamera*-kirjaston *Analyzer*-luokan toiminallisuutta, joka on esitelty ohjelmalistauksessa 1.

Sovelluksen arkkitehtuurissa erotettiin kuvanotto, neuroverkkojen laskenta ja tunnistustulosten raportointi omiksi rinnakkaisiksi osikseen (kuva 8). Kameran kuvanopeus voitiin näin pitää riippumattomana hahmontunnistuksen nopeudesta ja näin esikatselukuvaa voitiin näyttään tunnistusnopeutta suuremmalla kuvanopeudella. Sovelluksen arkkitehtuuri oli samanlainen kuin rinnakaistetussa testisovelluksessa, jossa kuvadata siirrettiin eri osien välillä jaetun muistin kautta. Toteutus tehtiin tukemaan sekä säikeisiin, että erillisiin prosesseihin pohjautuvaa rinnakkaisuutta.

Tunnistustulosten esittämistä varten toteutettiin yksinkertainen graafinen käyttöliittymä pythonin *tkinter*-kirjastolla. Käyttöliittymässä näytetään kameralta saatavaa kuvaa, neuroverkon laskennan tuloksia sekä laskennan hetkellistä ja keskimääräistä nopeutta. Lisäksi esitetään yleistä järjestelmän suorituskykyyn liittyvää informaatiota kuten keskusyksikön ja grafiikkayksikön lämpötilaa, sekä eri prosessoriytimien hetkellistä kuormitusta. Esimerkki käyttöliittymästä on esitetty kuvassa 9, jossa ajetaan kohteiden paikantamiseen suunniteltua neuroverkkoa kahdella kiihdyttimellä.



Kuva 9. *Konenäkösovelluksen käyttöliittymä.*

4.5 Käytetyt neuroverkkomallit

Testaukseen valittiin tyypillisiä valmiita neuroverkkorakenteita, jotka pystyttiin toteuttamaan Movidius kiihdyttimellä. Kaikki tässä työssä käytetyt verkot käännettiin *mvNCCompile*-kääntäjällä alun perin Caffe-neuroverkkoympäristöllä tehdyistä malleista. Tavoitteena oli vertailla eri tyyppisiä verkkoja, joiden ominaisuuksia ovat listattu taulukossa 3.

AlexNet [14] on kuvan luokitteluun suunniteltu neuroverkko. Vuonna 2012 se voitti ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) [19] kilpailun ensimmäisenä neuroverkkopohjaisena toteutuksena. Verkko on opetettu kyseisen kilpailun kuvatietokannalla luokittelemaan kuva johonkin 1000:sta määritellystä kategoriasta. Tässä työssä käytetty verkon Caffe-ympäristössä tehty malli on jaettu MNSC SDK kehitysympäristön mukana.

SqueezeNet [7] verkkototeutus on toinen työssä käytetty kuvan luokitteluun suunniteltu verkko, joka pystyy samaan luokittelutarkkuuteen kuin AlexNet, mutta sen suunnittelutavoitteena on ollut huomattavasti pienempi verkon parametrien määrää. Se on opetettu samalla tietokannalla kuin AlexNet. Verkon Caffe-malli on osa MNSC SDK:ta.

SSD-Mobilenet v1 on kohteiden paikantamiseen kuvasta suunniteltu verkko. Tämä toteutus on opetettu *PASCAL Visual Object Classes* [3] tietokannalla tunnistamaan ja paikantamaan kuvasta 20 erilaista kohdetta. Sen yhdistelmä kahdesta arkkitehtuurista. Verkon pääarkkitehtuuri ja toiminallisuus pohjautuu ”Single Shot Multibox Detector” (SSD) neuroverkkoon [15]. Mobilenet-verkot [6] ovat taas joukko verkkoja, joissa nopeutta on pyritty optimoimaan erityisesti rajoitettuihin laskentaympäristöihin kuten matkapuhelimiin. SSD-Mobilenet verkossa Mobilenet arkkitehtuuria käytetään optimoimaan alkuperäistä SSD toteututusta. Tämän verkon Caffe-malli on saatavilla ”Intel Movidius Neural Compute Application Zoo” (NC APP Zoo) hakemistosta[11].

	AlexNet	SQueezeNet	Mobilenet SSD
Lähde	MNSC SDK	MNSC SDK	NC App Zoo
Graafi tiedoston koko (Mt)	117,0	2,4	12,0
Parametrejä	$60{,}97 \times 10^6$	$1{,}25 \times 10^6$	$5{,}8 \times 10^6$
Summaavia kertolaskuja	$726{,}79 \times 10^6$	$832{,}6 \times 10^6$	1150×10^6
Vertailuja	$1{,}77 \times 10^6$	$9{,}61 \times 10^6$	$9{,}22 \times 10^6$
Yhteenlaskuja	$478{,}00 \times 10^3$	170×10^3	$9{,}26 \times 10^6$
Jakolaskuja	$955{,}05 \times 10^3$	1000	$9{,}26 \times 10^6$
Potenssiin korotuksia	$478{,}02 \times 10^3$	1000	$40{,}36 \times 10^3$

Taulukko 3. *Neuroverkkojen ominaisuuksia.*

5. SUORITUSKYKYANALYYSI

Laitteistolle tehtiin suorituskykymittauksia, joissa vertailtiin erilaisten ohjelmointiratkaisujen, neuroverkkojen ja kellotaajuuksien vaikutuksia. Tutkimuksessa mitattiin kuvan prosessointinopeutta, keskusyksikön kuormitusta ja lämpötilaa, sekä laitteiston virran- ja tehonkulutusta. Mittaukset tehtiin testisovelluksella, jossa etukäteen ladatut kuvat syötettiin neuroverkolle tietokoneen muistista sekä konenäkösovelluksella, joka prosessoi kameralta tulevia kuvia reaaliaikaisesti.

5.1 Testisovellus ilman kameraa

Ensimmäisessä vaiheessa tutkittiin neuroverkon laskennan nopeutta ja järjestelmän kuormitusta ilman kameraa tai käyttöliittymää. Mittauksissa käytettiin AlexNet neuroverkkoa, jolle syötettiin 227 x 227 x 3 kuvapikselin kokoista RGB-kuvaa eri kuvanopeuksilla. Testeissä etsittiin suurin nopeus, jolla verkon toteutus pystyi käsittelemään sille annetun kuvavirran. Testiohjelmistoja ajettiin Raspberry Pi:n *tehonsäästö* (engl. *powersave*) ja *suorituskyky* (engl. *performance*) tiloissa vastaavasti 600 MHz ja 1200 MHz kellotaajuudella. Jokainen mittaus tehtiin 5 minuutin mittaisessa ajossa, jonka aikana keskusyksikön lämpötila saavutti vakaan tilan. Mittauksen aikana Raspberry Pi oli koteloimattomana huoneenlämmössä (noin 22°C), varustettuna kuparisella jäädytyssiilillä.

Testiohjelmistosta oli 6 erilaista toteutusta. Kahdessa ensimmäisessä neuroverkon laskenta-algoritmi oli samassa säikeessä muun sovelluksen kanssa. Laskenta toteutettiin OpenCV kirjaston DNN moduulilla tai MNSC neuroverkkokiihdyttimellä. Loput neljä toteutusta käyttivät yhtä tai kahta MNCS-kiihdytintä ja verkon laskenta oli eriytetty kuvia syöttävästä sovelluksen osasta omaan erilliseen säikeeseen tai prosessiin. Mittaustulokset eri toteutusvaihtoehdoille löytyvät taulukoista 4 ja 5, jossa on listattu kussakin testiajossa saavutettu kuvanopeus, keskusyksikön lämpötila testin lopussa, prosessoriytimien yhteenlaskettu kuormitus, kiihdyttimille menevä datavirta sekä tehon- ja virrankulutus.

Sovellus	Kiihdytin	Nopeus k/s	Lämpötila °C	Kuorma %	Kuvadata Mbit/s	Teho W	Virta A
Yksi säie	OpenCV	0,7	61,8	93,6		2,6	0,5
	1	8,8	46,2	4,3	41,5	2,1	0,4
Monisäie, jaettu muisti	1	8,7	46,2	6,7	41,0	2,8	0,5
	2	17,6	48,3	13,5	83,0	4,33	0,84
Moniprosessi jaettu muisti	1	8,6	46,7	7,4	40,6	2,8	0,5
	2	17,3	48,3	14,5	81,3	4,33	0,84

Taulukko 4. *AlexNetin suorituskyky 600 MHz taajuudella.*

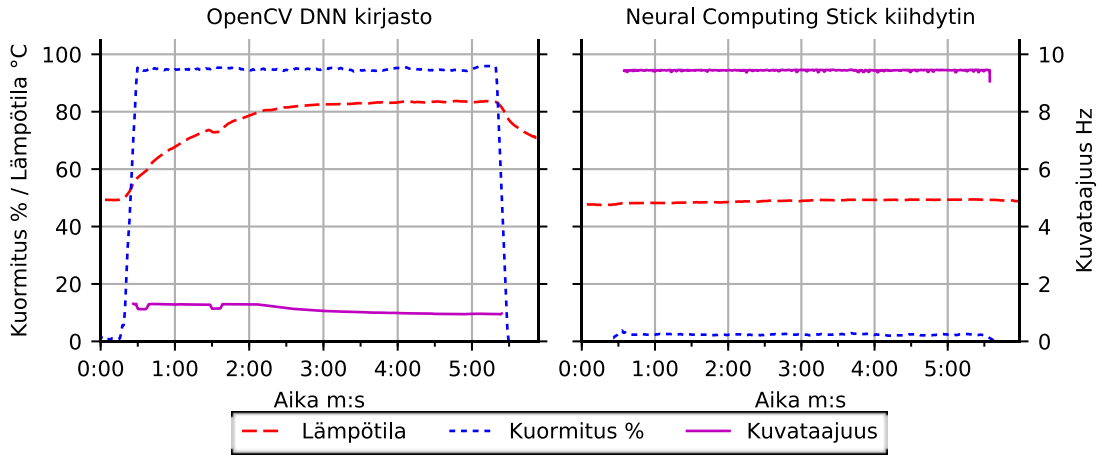
Sovellus	Kiihdytin	Nopeus k/s	Lämpötila °C	Kuorma %	Kuvadata Mbit/s	Teho W	Virta A
Yksi säie	OpenCV	1,0-1,3	83,8	94,9		3,6-4,7	0,7-0,9
	1	9,4	49,9	2,4	44,3	3,0	0,6
Monisäie jaettu muisti	1	9,3	51,0	4,1	43,9	3,0	0,6
	2	18,8	51,5	8,2	88,7	1,9*)	0,4*)
Moniprosessi jaettu muisti	1	9,3	51,4	4,7	43,9	3,1	0,6
	2	18,50	51,5	9,0	87,3	1,9*)	0,4*)

**) Kiihdyttimellä erillinen virransyöttö 0,4-1,0 A*

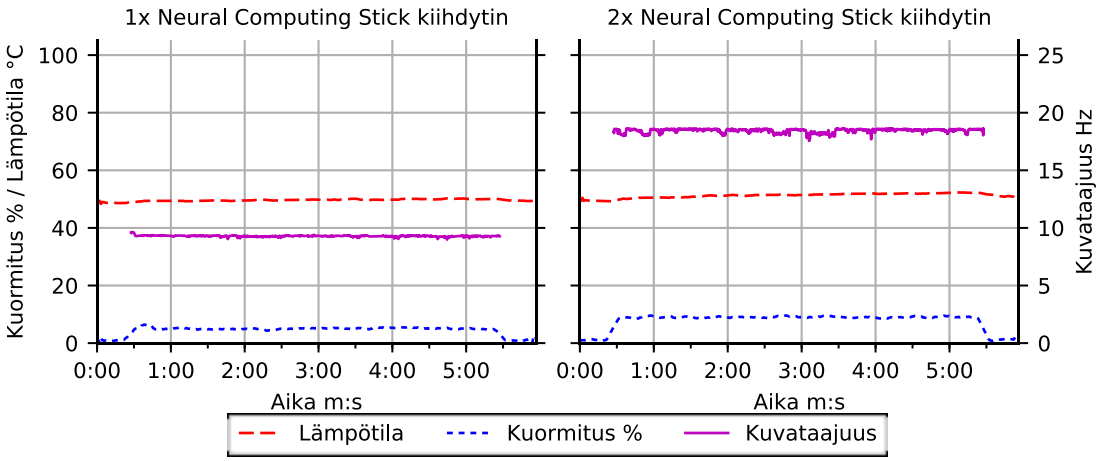
Taulukko 5. *AlexNetin suorituskyky 1200 MHz taajuudella.*

Pelkästään keskusyksikköä käyttävän OpenCV toteutuksen kuvanopeus 600 Mhz taajuudella oli 0,7 kuvaa sekunnissa. Paras 1,3 kuvan sekuntinopeus saavutettiin 1200 MHz taajuudella, kuitenkaan tätä nopeutta ei voitu ylläpitää jatkuvasti. Noin 1 minuutin ja 40 sekunnin ajon jälkeen prosessorin 95% kuormitus nosti keskusyksikön lämpötilan yli 80 °C. Tällöin järjestelmä ryhtyi automaattisesti vähentämään kellotaajuutta suojellakseen prosessoria ylikuumenemiselta. Testiajon aikana taajuus laski 900 MHz alapuolelle, jolloin lämpötila vakautui 83,8 °C tasolle ja saavutettiin lopullinen 0,95 kuvan nopeus. Testiajo on nähtävissä kuvassa 10. Kun vastaava testi ajettiin ilman jäähdytyssiiliä, prosessori saavutti 80 °C kriittisen rajan jo noin 40 sekunnissa ja kellotaajuus laski alle 800 MHz kuvanopeuden ollessa noin 0,9 kuvaa sekunnissa.

Vastaavassa testissä yhtä MNCS-kiihdytintä käyttävä toteutus saavutti 1200 MHz taajuudella nopeuden 9,4 kuvaa sekunnissa. Prosessorin kuormituksen oli tällöin 2,4% tasolla ja lämpötilan 50 °C alapuolella. Kiihdytintä käyttävä toteutus tässä testissä oli noin 10 kertaa nopeampi kuin ohjelmistopohjainen toteutus. 600 MHz taajuudella saavutettiin 8,8 kuvan sekuntinopeus, tällöin ero kiihdytetyn ja ohjelmistopohjaisen ratkaisun välillä on noin 12,6 kertainen.



Kuva 10. *AlexNet toteutuksien vertailu 1200 MHz taajuudella.*

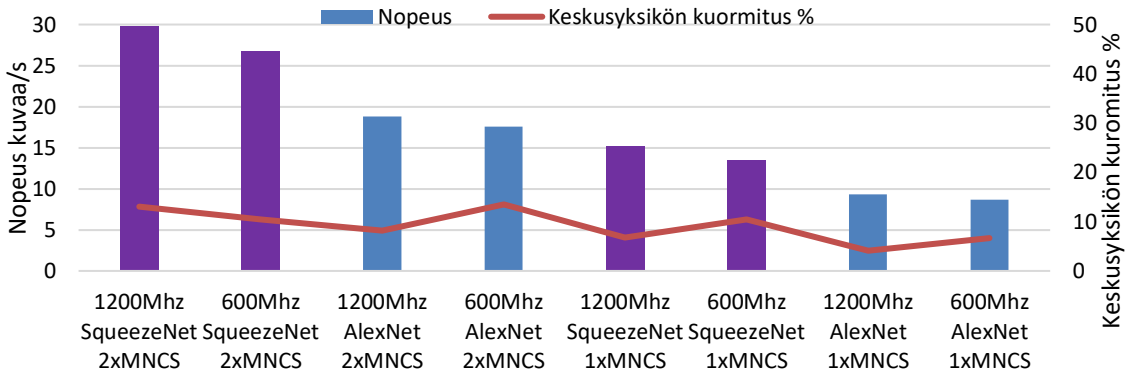


Kuva 11. *Vertailu 1 ja 2 kiihdyttimen suorituskyvystä AlexNetin laskennassa 1200Mhz taajuudella.*

Seuraavassa testissä vertailtiin yhden tai kahden kiihdyttimen käyttöä, sekä eri ohjelmistoratkaisujen vaikutusta suorituskykyyn. Toisin kuin edellisessä sovelluksessa näissä toteutuksissa verkon laskenta ja kuvien syöttö olivat rinnakkaisia ja kuvan siirto eri ohjelman osien välillä tapahtui jaetun muistin kautta. Verrattuna ensimmäiseen yksinkertaisempaan testisovellukseen prosessorin kuorma oli hieman suurempi, joka johtui lähinnä kuvalle tehdystä kopioinnista jaettuun muistiin. Kuitenkin yhdellä kiihdyttimellä saavuttiin käytännössä sama suoritusnopeus kuin ensimmäisessä testissä. Tämä oli odotettavissa, koska keskusyksikön kapasiteetti ei kummassakaan tapauksessa ole rajoittava tekijä. Kahden kiihdyttimen käyttö kaksinkertaistaa suoritusnopeuden ja prosessorin kuormituksen, joka pysyy kuitenkin vielä varsin matalalla tasolla. Kuva 11 vertailee 1 tai 2 kiihdyttimen käytön vaikutusta kuvataajuuteen ja keskusyksikön kuormitukseen.

Testeissä vertailtiin myös rinnakkaisprosessoinnin toteutusta erillisillä prosesseilla ja säikeistyksellä. Taulukoista 4 ja 5 nähdään, että näissä mittauksissa ei havaittu merkittäviä suorituskykyeroja toteutustapojen välillä. Tämä oli odotettavissa, koska kokonaiskuormitus on riittävän matalla tasolla ja toteutuksessa ei ole merkittävää määrää python tulkin ajamaa ohjelmakoodia, jota tarvitsisi jakaa useamman prosessoriytimen välillä.

Tehonkulutus eri toteutusvaihtojen välillä on suhteessa prosessorin kuormitukseen ja kiihdyttimien virrankulutukseen. Pienin noin 0,4 A virrankulutus saavutetaan yhden kiihdyttimen ratkaisulla 600 Mhz taajuudella. On huomattava, että jos 2 kiihdytintä on liitetty suoraan Raspberry Pi:hin saattaa järjestelmän tehonhallintayksikön jännite pudota satunnaisesti 4,65 V alapuolelle. Tällöin piirilevyllä olevan punaisen ledin sammuu lyhyeksi ajaksi. Kiihdyttimet toimivat vielä tässäkin tilanteessa, mutta keskusyksikön kellotaajuus säätyy tällöin hetkellisesti 600 MHz tasolle. Jos halutaan ylläpitää jatkuvaa 1200Mhz taajuutta, onkin tarpeen kytkeä kiihdyttimet erillisellä virtalähteellä varustettuun USB-keskittimeen. Tällöin Raspberry Pi:n virrankulutus pysyy suhteellisen matalalla 0,36 A tasolla, mutta kiihdyttimet tarvitsevat lähes 1 A hetkellisen virran.



Kuva 12. *Kiihdyttyjen kuvanluokittelu neuroverkkojen vertailu.*

5.2 Erilaisten neuroverkkojen vertailu

Seuraavissa testissä mitattiin SqueezeNet ja SSD-Mobilenet verkkojen suorituskykyä. Mittausasetelma oli samanlainen kuin luvun 5.1 testeissä. Kiihdytintä käytettäessä tulokset mitattiin säikeistetyllä jaettua muistia käyttävällä toteutuksella ja OpenCV tapauksessa samanlaisella yksisäikeisellä sovelluksella kuin aiemmissa testeissä. SqueezeNetin syöte oli sama kuin AlexNetillä, 227x277x3 kuvapikseliä. SSD Mobilenet käytti suurempaa 300x300x3 resoluution syötekuvaa. Mittaustulokset on listattu taulukoihin 6 ja 7.

SqueezeNet ja AlexNet-verkot on suunniteltu samaan kuvanluokittelutehtävään ja niiden suorituskyky vertaillaan kuvassa 12. Mittauksia vertaillessa nähdään, että kiihdytetty SqueezeNet on noin 1,6 kertaa nopeampi kuin AlexNet ja sen huippunopeus on lähes 30 kuvaa sekunnissa 1200Mhz kellotaajuudella. Prosessorin kuormitus ja lämpötila pysyvät tässä tilanteessa edelleen suhteellisen matalalla 13,1% ja 52,6 °C tasolla. Taulukoiduista tuloksista voidaan kuitenkin myös havaita, että ohjelmistopohjaisen toteutuksen suhteelliset erot ovat pienempiä kuin kiihdytystä käyttäessä ja näissä mittauksissa AlexNet on jopa hieman nopeampi.

Taajuus	Kiihdytin	Nopeus k/s	Lämpötila °C	Kuorma %	Kuvadata Mbit/s
600 MHz	OpenCV	0,58	60,1	93,0	
	1	13,5	44,6	10,5	63,7
	2	26,7	47,8	20,3	126,0
1200 MHz	OpenCV	1,0-1,2	81,2	93,4	
	1	15,1	48,85	6,8	71,2
	2	29,8	52,6	13,1	140,6

Taulukko 6. *SqueezeNetin neuroverkon suorituskyky.*

Taajuus	Kiihdytin	Nopeus k/s	Lämpötila °C	Kuorma %	Kuvadata Mbit/s
	OpenCV	0,4	58,5	88,6	
600 MHz	1	7,2	45,6	11,9	59,3
	2	14,6	48,3	22,9	120,3
	OpenCV	0,6-0,7	81,67	90,0	
1200 MHz	1	8,3	50,5	8,2	68,4
	2	16,4	53,7	15,9	135,1

Taulukko 7. *SSD-Mobilenet neuroverkon suorituskyky.*

SSD-mobilenet on kiihdytettynä nopeudeltaan samassa suuruusluokassa kuin AlexNet. Paras suoritusnopeus 16,4 kuvaa sekunnissa saavutetaan 1200Mhz kellotaajuudella, joka on noin 0,9 kertaa AlexNetin nopeus. OpenCV toteutus on taas suhteellisesti hitaampi, vastaavan vertailuluvun ollessa noin 0,6.

Kiihdytettyjen verkkojen suoritus analysoitiin tarkemmin myös MNCS SDK:n profilointityökalulla *mvNCProfile* ja tulokset on esitetty taulukossa 8. Profiloinnista nähdään, että verkon laskennan vaatima kiihdyttimessä ei vastaa täysin sovelluksen tasolta tehtyjä mittauksia. AlexNet on analyysin mukaan hitain verkko, mutta sovellustasolla sen nopeus on suurempi kuin SSD-Mobilenetin. Ero on selitettävissä SSD-Mobilenetin käyttämällä suuremmalla kuvaresoluutiolla, josta seuraa noin 1,5 kertainen datavirta kiihdyttimelle AlexNetiin verrattuna. Vastaavasti SqueezeNet on kiihdyttimen sisällä lähes 2 kertaa nopeampi kuin AlexNet, mutta suuremman kuvataajuuden aiheuttama datavirta vähentää sovellustasolla näkyvää nopeuseroa.

Mielenkiintoinen havainto on myös, että profilointitulos näyttää AlexNetille pienempää operaatiomäärä kuin muille verkoille, vaikka sillä on hitain suoritusaika. Tarkempi analyysi osoittaa, että AlexNetin suuret täysin kytketty neuroverkkokerrokset kuluttavat merkittävän osan laskenta-ajasta ja kuitenkin profilointi näyttää näille kerroksille hyvin pienen määrän liukulukuoperaatiota. Mahdollinen selitys tähän on se, kyseisten kerrosten toteutus ei ole sopiva Myriad prosessorin SHAVE yksiköille ja laskenta ei toteudu optimaalisesti.

	AlexNet	SqueezeNet	SSD Mobilenet
Suoritusaika (ms)	87,931	46,689	79,924
Suhteellinen suoritusaika	1,9	1,0	1,7
Liukulukuoperaatiot (kpl)	1332,5 x 10^6	1670,6 x 10^6	2297,1 x 10^6
Hitain kerros:			
suoritusaika (ms)	33,395	8,858	5,039
liukulukuoperaatiot (kpl)	0,018 x 10^6	347,7 x 10^6	209,7 x 10^6

Taulukko 8. *Neuroverkkojen profilointitulokset.*

5.3 Konenäkösovelluksen suorituskyky

Mitatun konenäkösovelluksen arkkitehtuuri on samankaltainen säikeistetyn testisovelluksen kanssa. Oleellisin ero on se, että kuvadataa saadaan reaaliaikaisesti kameralta vakionopeudella. Jos sovellus ei ehdi prosessoimaan kuvaa se ohitetaan. Tässä mittauksessa selvitettiin kuinka kameran käyttö vaikuttaa järjestelmän kuormitukseen ja toisaalta, miten sovelluksen vaatimat muut toiminallisuudet kuten samanaikaisesti päivittyvä käyttöliittymä ja erillinen etsinkuvan näyttö vaikuttavat suorituskykyyn. Mittauksessa käytettiin kahta kiihdytintä ja SqueezeNet verkkoa, koska sillä saatiin tässä työssä käytetyistä verkoista suurin kuvanopeus ja järjestelmän kuormitus. Tulokset löytyvät taulukosta 9.

Ensimmäisessä mittauksessa pyrittiin selvittämään kuinka pelkkä kameran käyttö vaikuttaa järjestelmän kuormitukseen. Tässä testissä sovellus ei näyttänyt kameran kuvaa näytöllä, eikä graafista käyttöliittymää. Kameran kuvataajuus asetettiin 30 kuvaan sekunnissa. Mittauksessa kiihdytetty neuroverkko pystyi analysoimaan kuvaa keskimäärin noin 29,6 nopeudella ja joutui ohittamaan vain 0,6% sisään tulevista kuvista. Saavutettu nopeus ja prosessorin kuormitus on käytännössä samalla tasolla kuin testisovelluksessa. Ainut oleellinen ero on hieman suurempi prosessorin lämpötila 57°C, joka on seurausta kameradataa käsittelevän ”Video Core”-yksikön toiminnasta.

Seuraavassa vaiheessa lisättiin järjestelmän kuormitusta näyttämällä esikatselukuvaa Rapspberry Pi:hin kytkettyyn teräväpiirtomonitorilla. Tämä ei lisää oleellisesti keskusyksikön kuormitusta, koska kuvadata siirtyy suoraan ”Video Core”-yksikön sisällä näyttöpuskuriin. Näin ollen kuvanopeus pysyi edellisen mittauksen tasolla, mutta järjestelmän lämpötila nousi hieman 61,2 °C.

Tämän jälkeen sovellukseen lisättiin käyttöliittymä, joka raportoi verkon tuloksia ja näytti sisäänmenokuvaa sovellusikkunassa, joka näytettiin eri tietokoneessa verkkoyhteyden yli käyttäen X-ikkunointiprotokollaa. Tästä seurasi prosessorikuormituksen nousu 26,3% tasolle. Tällöin ohitettujen kuvien määrä nousi 3,4 prosenttiin ja vastaava keskimääräinen kuvataajuus jäi 28,9 kuvaan sekunnissa.

Sovellus	Tavoite nopeus k/s	Saavutettu nopeus k/s	Hukattuja kuvia %	Lämpötila °C	Kuorma %
Kamera	30	29,64	0,6	57,0	13,4
+ Etsinkuva	30	29,64	0,7	61,2	13,6
+ Käyttöliittymä	30	28,9	3,4	63,0	26,3
+ 25% lisäkuorma	30	28,3	5,7	69,3	50,5
	30	26,8	11,5	77,3	74,8
+ 50% lisäkuorma	25	24,6	1,0	78,4	72,6
	15	15,0	0,0	75,2	67,2

Taulukko 9. *Konenäkösovellus SqueezeNet 1200Mhz 2 kiihdytintä.*

Viimeisessä vaiheessa testattiin järjestelmän toimintaa, kun konenäkösovelluksen rinnalle lisätään muita kuormittavia prosesseja, ensin 25% lisäkuorma ja sitten 50% lisäkuorma. Mittauksista nähdään, että kokonaiskuormituksen ollessa noin 75% tasolla hukattujen kuvien määrä on jo 11,5 % ja järjestelmän lämpötila lähentelee kriittistä 80 °C rajaa. Pienentämällä kuvanopeutta voidaan kuitenkin tässäkin tilanteessa saavuttaa hyvä suorituskyky. 25 kuvan sekuntinopeudelle päästään 1% hukattujen kuvien määrään ja 15 kuvan nopeudella kaikki kamerakuvat voidaan käsitellä ilman ongelmia.

5.4 Suorituskykyyn vaikuttavia tekijöitä

Tässä luvussa mitattiin neuroverkkojen suorituskykyä tietyssä laite- ja ohjelmistoympäristössä. Testit ovat suhteellisen suppeita, mutta niitä voidaan pitää suuntaa antavina, kun arvioidaan käytetyn järjestelmän soveltumista muihin saman kompleksisuuden konenäkötehtäviin. Käytännön sovellukset ovat kuitenkin usein hieman erilaisia. Seuraavassa esitetään joitain huomiota, jotka voivat vaikuttaa suorituskykyyn eri tilanteissa.

Luvussa 5.2 tehdyssä vertailussa huomattiin, että eri verkkoarkkitehtuurien suhteelliset suorituskykyerot ovat erilaisia ohjelmistopohjaisilla ja kiihdytetyillä toteutuksilla. Myös kiihdytettyjen verkkojen profilointi osoitti, että laskenta saattaa jakautua epäoptimaalisesti käytetyn arkkitehtuurin sisällä. Onkin suositeltavaa, että sovellusta kehitettäessä neuroverkon suoritus profiloidaan toteutusympäristössä ja sitä muutetaan tarvittaessa paremmin ympäristöön sopivaksi. Kun verkko toteutetaan kiihdyttimessä, tehdään käännös sen alkuperäisestä esitysmuodosta, esimerkiksi TensorFlow- tai Caffe-mallista, laitteiston omaan esitysmuotoon ja ohjelmakoodiin. On mahdollista, että tämä käännösprosessi tukee jotain tiettyä verkon esitystapaa paremmin kuin toista. Jos käytetty verkkomallista on tarjolla useita esitysmuotoja kannattaa näiden välillä tehdä suorituskykyvertailu.

Luvussa 5.3 testatun sovelluksen kokonaiskuormitus oli suhteellisen pieni. Suurin vaikuttava tekijä tähän on se, että sovellus hyödyntää ”Video Core”-yksikköä kuvan esikäsitellyssä, erityisesti kameraprosessointiin ja kuvan koon muuttamisessa. Tämän seurauksena keskusyksikön ei tarvitse käsitellä suurikokoisia kuvia ja USB väylän siirtonopeudesta ei tule rajoituksia. Jos sovellus tarvitsee merkittävästi suurempaa resoluutiota tai monimutkaista keskusyksiköllä tehtyä kuvan esikäsittelyä saattaa prosessorin kapasiteetti muodostua pullonkaulaksi. Myös jos järjestelmässä on suuria määriä samanaikaista USB tai verkkoliityntää käyttävää dataliikennettä tämä saattaa rajoittaa kommunikointia kiihdyttimien kanssa.

Tehdyissä testeissä laitteisto oli koteloimaton. Suljetussa kotelossa järjestelmän lämpötilarajoitukset tulevat vastaan pienemillä kuormilla ja lyhyimmillä prosessointiajoilla. Lämpötilarajoitus ei koske pelkästään Raspberry Pi laitteistoa, myös neuroverkkokiihdyttimet laskevat suoritusnopeutta, jos lämpötilarajat ylittyvät.

6. YHTEENVETO

Työn tavoitteena oli selvittää miten Raspberry Pi:n ja Movidius Neural Computing Stick soveltuvat neuroverkkoja käyttävän konenäkösovelluksen toteuttamiseen. Työssä tutkittiin järjestelmän suorituskykyä mittaamalla tehonkulutusta, prosessointinopeutta ja kuormituksesta erilaisissa käyttötilanteissa. Kehitystyön sujuvuutta tässä ympäristössä testattiin käytännössä toteuttamalla sovellus kahteen tyypilliseen konenäköongelmaan, kuvan luokitteluun ja kohteiden paikantamiseen kuvasta.

Mittausten perusteella voidaan todeta, että työssä käytetyllä järjestelmällä saavutetaan riittävä suorituskyky, jotta kameran kuvadataa voidaan prosessoida reaaliaikaisesti suhteellisen suurilla kuvanopeuksilla. Tämä antaa mahdollisuuksia kehittää ja testata sovelluksia, joissa tavoitellaan nopeaa reaaliaikaista vastetta. Tällaisia ovat esimerkiksi jonkin teollisen laitteen tai prosessin toiminnan ohjaus konenäköjärjestelmällä. Toinen esimerkki on konenäön käyttäminen osana käyttöliittymää. Kiihdytystä käytettäessä jää vielä prosessointikapasiteettia keskusyksikölle, jolloin sovelluksen muita toiminallisuuksia on myös helppo tutkia ja kehittää samalla samassa ympäristössä. Raspberry Pi tarjoaa monipuoliset fyysiset liitynnät ulkomaailmaan ja siihen on mahdollista liittää muita laitteita osaksi kokonaista konenäköjärjestelmää. Laitteiston on pienikokoinen ja tehonkulutus on riittävän matalalla tasolla, joten tarvittaessa siitä voidaan tehdä suhteellisen kompakti akuilla toimiva kannettava laite. Tämä tekee sovelluksen helpon testaamisen erilaisissa käyttöympäristöissä mahdolliseksi.

Sovellusta kehitystyön aikana tehtiin subjektiivisia havaintoja ohjelmistoympäristön helppokäyttöisyydestä ja työn tuottavuudesta. Tässä työssä käytetty python ohjelmointikieli on erityisen sopiva iteratiiviseen ohjelmistokehitykseen, koska tulkattuna kielenä aikaa ei kulu käännösprosessiin ja näin erilaisten toteutusratkaisujen nopea kehitys ja testaaminen. Työssä käytetyt keskeiset ohjelmistokirjastot ovat hyvin optimoituja ja suoraviivaisia ottaa käyttöön. Kirjastojen ohjelmistorajapinnat ovat hyvin dokumentoituja ja toiminnallisuudesta löytyy riittävästi ohjelmointiesimerkkejä. On syytä huomioida myös, että Raspberry Pi on täysverinen suhteellisen tehokas tietokonejärjestelmä. Linux käyttöjärjestelmään on tarjolla runsaasti sovelluksia, joita kehitystyössä voidaan hyödyntää. Tässä työssä kaikki ohjelmistonkehitys ja osa dokumentaatiosta tehtiinkin suoraa Raspberry tietokoneessa. Tutkimuksen aikana kehitystyön haasteet ja rajoitukset syntyivät pääsääasiassa erilaisten neuroverkkojen sovittamisesta neuroverkkokiihdyttimeen. Koska alan tutkimus on nopeaa ja tuottaa uusia ratkaisuja nopeasti eivät kaikki uusien verkkojen tarvitsemat ominaisuudet ole saatavilla kiihdytetyssä ympäristössä.

Yhteenvetona voidaan todeta, että työssä tutkittu järjestelmä, joka yhdistää pienoistietokoneen ja erillisen neroverkkokiihdyttimen, soveltuu hyvin konenäköön liittyvään tutkimus- ja tuotekehitystyön.

LÄHTEET

[1] Berkeley AI Research. Caffe neuroverkkoympäristön dokumentaatio. Sivusto. Saatavissa: http://caffe.berkeleyvision.org/

[2] DeepMind Google Alphabet. Alpha Go. Saatavissa: https://deepmind.com/research/alphago/

[3] Mark Everingham et al. The PASCAL Visual Object Classes Challenge: A Retrospective. International Journal of Computer Vision 111 s. 98-136. Springer 2015.

[4] Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning. MIT Press, 2016. Saatavissa: http://www.deeplearningbook.org

[5] Google. Do it yourself artificial Intelligence. Sivusto. Saatavissa https://aiyprojects.withgoogle.com/

[6] Andrew G. Howard et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. Saatavissa: http://arxiv.org/abs/1704.04861

[7] Forrest N. Iandola et al. Squeezenet: alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. 2016. Saatavissa: https://arxiv.org/abs/1602.07360v4

[8] Intel. OpenVino Toolkit ympäristön dokumentaatio. Sivusto. Saatavissa: https://software.intel.com/en-us/openvino-toolkit.

[9] Intel. VPU Product Brief. 2016. Saatavissa: https://movidius-uploads.s3.amazonaws.com/1532512604-1503680554-2016-12-12_VPU_ProductBrief.pdf

[10] Intel. The Intel® Movidius™ Neural Compute SDK. 2018. Sivusto. Saatavissa: https://movidius.github.io/ncsdk/index.html.

[11] Intel. The Intel® Movidius™ Neural Compute App Zoo. 2018. Sivusto. Saatavissa: https://github.com/movidius/ncappzoo

[12] Hsu Feng-hsiung, Campbell Murray. Deep Blue System Overview. Proceedings of the 9th international conference on Supercomputing. s. 240–244. ACM 1995.

[13] Keras neuroverkkoympäristön dokumentaatio. Sivusto. Saatavissa: https://keras.io/

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, s. 1097-1105. 2012.

[15] Liu. W. et. al. SSD: Single Shot MultiBox Detector. Computer Vision – ECCV 2016. Lecture Notes in Computer Science, vol 9905. Springer. 2016.

[16] Microsoft. The Microsoft Cognitive Toolkit ympäristön dokumentaatio. Sivusto. Saatavissa: https://docs.microsoft.com/en-us/cognitive-toolkit/

[17] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015. Saatavissa: http://neuralnetworksanddeeplearning.com/index.html

[18] OpenCV konenäköohjelmiston dokumentaatio. Sivusto. Saatavissa: https://opencv.org/

[19] Olga Russakovsky, Jia Deng et al. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision 115 s.211-252. 2015. Saatavissa: https://doi.org/10.1007/s11263-015-0816-y

[20] Raspberry Pi Foundation. Raspberry PI dokumentaatio. Sivusto. Saatavissa: https://www.raspberrypi.org/documentation/

[21] Rumelhart D., Hinton G., and Williams R. Learning representations by back-propagating errors. Nature 1986, 323, s. 533–536.

[22] TensorFlow neuroverkkoympäristön dokumentaatio. Sivusto. Saatavissa https://www.tensorflow.org/

[23] Theano laskentaohjelmiston dokumentaatio. Sivusto. Saatavissa: http://deeplearning.net/software/theano/